**CISC 220 – 010 Homework #3 Due Monday, September 25, 2000**

For this week's homework, we will working with Big-O notation, add another feature to the `String` class, and code the methods for the `ArrayLinearList` class.

**Part A)** The `String` class we just worked on for homework is actually missing some potentially useful features. For example, it would be nice if we could concatenate strings, as shown in the following code.

```
int main() {

   String A("Hi,"),  B("how are you?")'

   String C = A + B;

   cout << C << end;

   // should output "Hi, how are you?"

}
```

Implement the overloaded + operator to make this happen. Make a small driver to test the new feature, then make a script file (`HW-3-A.scr`) in which you cat the header file and implementation file for the `String` class, the driver, and then compile and run the driver.

**Part B)** Give a brief analysis, showing that the Bubble sort we have implemented has worse case running time of $O(n^2)$. Write a driver in whom you make an array of size N, store values in the array, which will cause a worse case for the bubble sort, and then invoke bubble sort on this array. Then run the program for values of `N` of 10, 100, 1000, 5000, and 10000, and make a table showing the value of N, the actual running time, and an estimate of the constant "c". To get the running time of a program on Unix, simply run it this form:

```
%  time  a.out
```

and note the first time value – which is the total CPU time for that program run. The values of "c" should be fairly constant – are they?

Make a script file (`HW-3-B.scr`) containing a cat of your driver, and runs for the five values of N. Print this script file, and write your table on it for hand in.

**Part C)** Here is the class definition for the static array implementation of the linear list:

```
//Dynamic Array Linear  List header file

template <class ListEntry>
class ArrayLinearList {
   public:
      List(unsigned long int MaxS = 50);
      ~List();
      void Clear();
      bool empty() const;
      bool full() const;
      int size() const;
      int insert( int pos, const ListEntry& x );
      int remove( int pos, ListEntry& x);
      int retrieve( int pos, ListEntry& x ) const;
      int replace( int pos, const ListEntry& x );
      void traverse( void (*visit)(ListEntry& x));
   private:
      unsigned long int Size;
      unsigned long int MaxSize;
      ListEntry* Data;
};
```

(Remember that we are working with a list whose positions are 1, 2,..., N - *NOT* 0, 1, ..., N-1.) Finish the header file by coding all of the member functions. Don't forget: since this is a template class, everything goes in the header file. Then write a driver to test the functions. First make a list of 10 integers, insert, remove, replace and retrieve some, and print the resulting list. Once your code works, make a script file (`HW-3-C.scr`) containing a cat of the header file and driver, and the sample run.

**Extra Credit)** Using the list class in Part C) implement a "Sieve of Erostothenes" for finding prime numbers. Do this as follows: 1) make a List of ints of size 1000, and put the integers 2 through 1001 in it; 2) starting with the first entry in the list (2 in this case), retrieve each entry in turn, and remove all that are multiples of 2; 3) Now take the 2nd entry in the list (3) and remove all multiple of it (using only list functions retrieve and remove); 4) then do the same for the 3rd entry (5), and so forth until you get to the end of the list. A print out of the list at this point should show all primes. Once you get this algorithm to work make a script file (`HW-3-EX.scr`) in which you cat the program and a run showing the resulting primes.

**What to hand in**: Hand in the three script files generated above (and the extra credit if you did it), stapled together as one set. (Don't forget to print your name on the front page!)